Otimizações Importantes para Paralelismo OU em Prolog sobre Máquinas com Memória Distribuída

Cláudio Geyer, Jacques Briat, Michel Favre, J. Chassin de Kergommeaux*

Projet CMaP, Laboratoire de Génie Informatique - IMAG
BP 53 X,
38041 Grenoble Cedex
França

Resumo

Este artigo descreve novas otimizações adotadas no modelo Opera ([3]) de execução do paralelismo OU multi-seqüencial em Prolog, para máquinas com memória distribuída. Uma técnica especial para a compilação das variáveis condicionais, baseada na máquina abstrata de Warren, é descrita, reduzindo o número de ligações profundas. Um novo método para a implementação eficiente da instalação de tarefas, baseado na cópia de contextos, é apresentado. A cópia é executada em paralelo ao trabalho da WAM, exigindo a inclusão do valor da ligação na pilha de registro de ligações. Esse método consome menos memória e diminue a complexidade da cópia incremental. Uma técnica é proposta para a implementação do predicado de corte (*cut*), mantendo-se a semântica original de Prolog, sendo mais simples e consumindo menos memória que a de [1]. Uma arquitetura hierárquica do sistema é proposta para as configurações com grande número de processadores (> 100) da máquina Supernode, sobre a qual foi implantado o protótipo Opera.

Palavras-chaves: paralelismo OU, Prolog, memória distribuída, otimizações.

1. Introdução

Este artigo descreve algumas otimizações importantes adotadas no modelo Opera de execução do paralelismo OU em Prolog, para máquinas com memória distribuída. Essas técnicas não foram abordadas em artigos anteriores sobre o modelo Opera ([3]), por motivos diversos como falta de espaço e, principalmente, uma evolução após a redação daqueles artigos.

O modelo Opera, para a execução do paralelismo OU em Prolog, foi concebido dentro dos estudos desenvolvidos para a obtenção de aumentos de desempenho da linguagem Prolog (equipe Flop, em Grenoble). Tais estudos estão centrados principalmente na exploração do paralelismo. A maior simplicidade do paralelismo OU foi a principal razão para a sua escolha como objetivo inicial, em detrimento do paralelismo E ([7]).

* Atualmente na Universidade Federal do RGS, Instituto de Informática.
Av. Bento Gonçalves, 9500 - Bloco IV - Bairro Agronomia
Caixa Postal 15064 - 91501 - Porto Alegre/RS - Brasil
N. Fax: (55) (051) 336-5576 - E-mail: geyer@inf.ufrgs.br

Desde o início, optou-se pelo paralelismo implícito, sem alteração de Prolog, o que permite a execução direta de aplicações existentes, desenvolvidas em Prolog. Essa abordagem também mantem as qualidades de linguagem declarativa de Prolog, permitindo uma programação paralela de mesma complexidade que a seqüencial ([7]).

A arquitetura alvo de Opera é a classe das máquinas com memória distribuída, devido a essas máquinas serem novidade e apresentarem um potencial de maior quantidade de processadores.

Considerando-se, como hipótese, que as máquinas de memória distribuída não são (ainda) próprias para comunicação muito freqüente, o modelo Opera foi concebido como um modelo OU multi-seqüencial. Tais modelos apresentam maior granularidade que os modelos OU de cláusula, pois uma tarefa é definida como uma resolvante completa (cláusula e continuação). Como modelo multi-seqüencial, Opera tem as seguintes características principais:

- Opera é constituído de N máquinas abstratas seqüenciais Prolog (MAP), onde N é função dos recursos em memória e processador da máquina alvo;
- cada MAP executa uma interpretação procedural clássica, criando nós OU e fazendo retrocessos quando ela atinge uma folha da árvore OU;
- no início da execução, todas as MAP estão inativas, uma só recebendo a resolvante inicial e tornando-se ativa;
- cada MAP inativa torna-se ativa pela aquisição de tarefas suspensas armazenadas nos nós OU das máquinas ativas;
- uma MAP ativa torna-se inativa quando, chegando a uma folha da árvore OU, ela não possuir alternativas suspensas nos seus nós OU;
- a execução termina quando todas as máquinas tornam-se inativas.

A gestão dos contextos múltiplos é feita por duplicação ([2]), e não por compartilhamento de dados como na maioria dos modelos encontrados na literatura. Tal característica implica, quando da migração de uma tarefa de máquina ativa para inativa, na transferência (cópia) do contexto relativo ao nó OU ao qual pertence a tarefa. Essa cópia é justificada, na ausência de uma memória comum, pelo relativamente alto custo de um acesso não-local em máquinas com memória distribuída, e pelas características dos acessos de uma MAP: freqüentes e a células pequenas (por exemplo, palavra de memória). Além disto, a operação de cópia pode usufruir do recurso de comunicação por DMA (em paralelo ao cálculo), existente nas máquinas paralelas em geral. Essa cópia de contexto é efetuada de modo incremental, isto é, as seções do contexto a ser transferido, já existentes na máquina inativa, não são transferidas ([2]).

Em Opera, adotou-se a compilação como método de implementação de Prolog. Uma implementação seqüencial ineficaz, por exemplo por interpretação, poderia conduzir a ganhos artificiais no desempenho do sistema paralelo, pois o custo da comunicação seria relativamente dimuído pelo real aumento do tempo de execução seqüencial. A MAP de Opera é baseada na máquina abstrata de Warren (WAM [6]), a mais utilizada em implementações de Prolog.

Um protótipo desse modelo foi implementado sobre uma máquina Supernode, baseada em Transputers, com memória distribuída ([3, 4]). Tal protótipo apresentou bons ganhos de desempenho, como por exemplo 14 para o programa queens(10) sobre 16 processadores. No

entanto, durante o desenvolvimento e avaliação do protótipo, diversas otimizações foram concebidas. Algumas dessas otimizações, decritas nesse artigo, são:

- conceito de variáveis condicionais aplicado à compilação;
- novo método de instalação de uma tarefa, baseado em cópia de contexto;
- implementação do operador de corte;
- arquitetura de processos para escalonamento e protocolo de comunicação para configurações do Supernode com mais de 100 processadores.

Uma análise preliminar das duas primeiras otimizações indicam a possibilidade de maiores ganhos de desempenho. A terceira, ou operador de corte, é necessária para a execução de muitos programas Prolog reais, e a quarta é exigida pela característica hierárquica das configurações maiores do Supernode.

2. Variáveis Condicionais

Esta seção descreve uma proposição de compilação para determinadas variáveis WAM, aqui denominadas condicionais ([4]). O objetivo é a redução do custo extra gerado por essas variáveis na execução da operação de cópia de contextos.

A compilação de Prolog em um programa WAM mapeia as variáveis Prolog em diferentes tipos de variáveis WAM. As variáveis WAM são classificadas em ([6]):

- locais: não são referenciadas em estruturas, sendo alocada para cada uma somente uma célula no registro de ativação (um por cláusula) correspondente da pilha local;
- globais: ao menos uma das referências é feita em uma estrutura, sendo que, em caso de modo de escrita da estrutura, uma célula é alocada na pilha global, além da célula da pilha local.

A alocação e a inicialização de uma célula de variável na pilha local são realizadas por intruções distintas, podendo ocorrer entre as duas a criação de um nó OU. Tal fenômeno não ocorre com as células da pilha global.

Uma **ligação condicional** de uma variável WAM é aquela que é separada da alocação da variável por um ou mais nós OU. Quando de uma cópia de contexto, uma ligação condicional deve ser desfeita pela WAM inativa, caso a tarefa exportada corresponda a um dos nós existentes entre a alocação da variável e a ligação. Tal operação exigiria a transferência de toda a pilha de registro de ligações e a suspensão momentânea do trabalho da WAM ativa. Portanto, uma cópia simples de contexto implica uma forte sincronização com o trabalho normal da WAM, e a transferência e o exame de registros de ligações não-necessárias à WAM inativa.

Para evitar esses dois problemas, o modelo Opera introduziu uma pilha de variáveis e a datação (profundidade dos nós OU). A cada ligação condicional, o valor e a data de ligação são registradas nessa pilha adicional numa célula alocada com a inicialização da variável. A cópia de contexto transfere essa pilha, e a máquina inativa desfaz as ligações examinando a data de cada uma, prescindindo da pilha de registro de ligações ([3]).

O uso da cópia incremental introduz um problema de coerência com relação às variáveis locais condicionais. Uma variável condicional é copiada para uma máquina inativa em estado ainda não-inicializado, caso a cópia seja realizada entre a alocação e a inicialização da variável (o nó OU da alternativa transferida foi criado entre a alocação e a inicialização). Caso a alternativa da máquina inativa falhar antes da inicialização e essa máquina importar uma alternativa posterior à inicialização, a variável em questão restará não-inicializada, pois a mesma não será novamente copiada. Uma variável não-inicializada implica erros de execução, caso seja usada em determinadas instruções da WAM.

A solução para evitar esse problema é simples. As variáveis condicionais locais, cuja primeira ocorrência encontra-se após o primeiro objetivo (*goal*), deverão ser inicializadas logo após as instruções geradas para a cabeça da cláusula. Para tal, em Opera, uma nova instrução *init_perm(X)* é gerada para cada uma dessas variáveis, e a primeira ocorrência é compilada como *value* (leitura da variável sem inicialização), e não como *variable*. Essa inicialização "avançada" implica um sobre-custo em relação ao procedimento normal da WAM, pois em caso de falha em algum objetivo da cláusula um maior número de incializações se tornará inútil.

A pilha de variáveis e a datação (PVD) introduzem um sobre-custo constante nas operações de alocação, inicialização e ligação das variáveis. Com o intuito de reduzir esse sobre-custo, todas as ligações incondicionais devem ser efetuadas de modo **superficial**, isto é, sem uso do mecanismo PVD. Determinar à compilação a característica incondicional condicional de uma ligação é um problema complexo, o que implicaria uma implementação totalmente dinâmica para essa otimização.

Por outro lado, é possível classificar as variáveis em **incondicionais**, aquelas que certamente terão ligações incondicionais, e **condicionais**, aquelas que eventualmente terão uma ligação condicional. A diferenciação na compilação desses dois tipos de variáveis é simples, pois elas já são normalmente compiladas de modo diferente na WAM.

As variáveis incondicionais são as variáveis locais aparecendo na cabeça da cláusula. Elas serão certamente ligadas a um argumento da chamada do próprio predicado, podendo usufruir do mecanismo de ligação superficial. As variáveis condicionais são as outras variáveis locais e todas as globais. Para as variáveis condicionais locais, torna-se obrigatório a alocação e inicialização da célula na pilha de variáveis quando da inicialização pela *init_perm*. A cada ligação, verifica-se o carácter incondicional condicional e, no primeiro caso, efetua-se uma ligação superficial.

As variáveis globais cuja primeira ocorrência é no corpo da cláusula exigem uma inicialização com alocação de célula na pilha de variáveis, efetuadas ao mesmo tempo quando dessa ocorrência. Quanto às variáveis globais que ocorrem na cabeça da cláusula, propõe-se uma compilação diferenciada, pois existe a possibilidade de que elas não sejam ligadas por essas ocorrências. A última ocorrência, anterior à primeira chamada (até o primeiro objetivo inclusive), é compilada com versões especiais das respectivas instruções WAM. Essas versões alocam e inicializam uma célula na pilha de variáveis, caso a variável ainda esteja livre e caso ele seja criada (modo gravação para *unify_variable*, *unify_local_value* e *unify_value*).

3. Instalação de Tarefas

A instalação de uma tarefa consiste de três etapas:

- preparação dos dados a enviar, com atualização dos estado da WAM exportadora (pilha de nós OU);
- envio e recepção das seções (não-comuns) das pilhas necessárias à execução da tarefa;
- restauração do estado da WAM importadora (registradores e ligações condicionais).

O método proposto anteriormente em [3], baseado na pilha de variáveis e na datação, apresenta alguns inconvenientes quando acoplado à cópia incremental e à transferência de alternativas pertencentes à mais de um nó OU.

A cópia incremental elimina a transferência das seções comuns às duas WAM, mas não exclue a restauração das ligações condicionais sobre as variáveis dessa seção. Em outros termos, as ligações efetuadas pela WAM importadora, após o nó OU mais recente entre os com alternativas transferidas, devem ser desfeitas, assim como as ligações efetuadas pela WAM exportadora, antes do mesmo nó, devem ser instaladas na WAM importadora.

Duas alternativas gerais são possíveis para efetuar-se a instalação das ligações da WAM exportadora:

- envio completo da pilha de variáveis, isto é, inclusive a seção comum. Essa alternativa desfaz também as ligações da WAM importadora, mas ela inclue ligações já existentes na WAM importadora (redundância de dados);
- determinação dessas ligações pela WAM exportadora (por exemplo, percorrendo a seção comum da pilha de variáveis e analisando a data de ligação), com conseqüente envio a WAM importadora.

A transferência de alternativas pertencentes a vários nós OU exige a cópia de determinadas seções da pilha de registro de ligações. Essas seções, situadas entre aqueles nós, serão necessárias quando de retrocessos (locais) da WAM importadora. No entanto, essas seções e as seções transferidas da pilha de variáveis contem informações redundantes.

Assim, com o intuito de reduzir os sobre-custos acima do método PVD, propõe-se um outro método para a instalação de tarefas, baseado em uma pilha de variáveis mais simples e em uma modificação da pilha de registro de ligações (PVV [4]). A pilha de variáveis contem somente o valor da ligação, excluindo-se a data de ligação. A pilha de registro das ligações contem, além do endereço da variável, o valor da ligação.

O princípio geral desse método é a transferência exclusiva das ligações válidas à WAM importadora. Essa validade é determinada pelo mais recente nó OU entre os transferidos.

A instalação é realizada com as seguintes etapas:

- os nós OU são transferidos, enquanto a WAM importadora desfaz as ligações efetuadas por ela própria após o nó OU comum entre as duas WAM;

- o nó OU comum e o nó OU transferido mais recente são utilizados para a determinação das seções não comuns das pilhas;
- a seção não-comum da pilha de registro de ligações é transferida;
- as seções não-comuns das pilhas global e local são transferidas, enquanto a WAM importadora instala na pilha de variáveis as ligações existentes na seção não-comum da pilha de registro de ligações.

Esse método apresenta uma sincronização de condição, entre a recepção e o trabalho de instalação efetuados pela WAM importadora: a recepção da pilha de registro de ligações somente pode começar após o fim da desligação das ligações anteriores ao nó OU comum. O tempo de retardo corresponde à diferença entre o fim da recepção dos nós OU e o fim da desligação, podendo então ser nulo.

Uma etapa adicional, para a inicialização da seção não-comum da pilha de variáveis, é necessária caso a mesma não seja estática (total, no início da execução).

Uma comparação preliminar entre esse método e o PVD (com percurso da seção comum da pilha de variáveis para determinação das ligações a transferir) resulta em:

I1 - diferença na quantidade de dados a transferir: 2 vezes (2 células) o número de variáveis condicionais da seção não-comum (PVD - pilha de variáveis) contra o número de ligações efetuadas durante a seção não-comum (PVV - pilha de registro de ligações contem uma célula a mais);

I2 - diferença no cálculo (iterações sobre as pilhas): número de variáveis condicionais da seção comum (PVD) contra o número de ligações condicionais efetuadas durante a seção não-comum (PVV);

I3 - espaço (alocação da segunda célula de ligação condicional: data ou valor na pilha de registro de ligações): possibilidade de alocação sem que a ligação ocorra (PVD) contra alocação somente em caso de ocorrência de ligação (PVV).

Conclusões definitivas quanto aos dois primeiros itens (I1 e I2) exigem medidas experimentais sobre um número representativo de programas Prolog, no sentido de avaliar-se as quantidades médias de variáveis condicionais não-comuns e comuns, e de ligações condicionais efetuadas durante a seção não-comum. Na falta dessas medidas, o método PVV parece mais indicado devido ao fator 2 no primeiro item (I1) do método PVD, ao fato de que numa mesma seção o número de ligações é no máximo igual ao número de variáveis e ao item I3.

4. Operador de Corte (cut)

O método proposto para a implementação do operador de corte (*cut*) no modelo Opera é um método de marcação dos nós OU, dependentes de um corte em potencial, com conseqüente inibição do paralelismo ([4]). Um método eficaz deveria evitar a penalização de cláusulas sem cortes, o que é difícil devido a falhas ocorridas antes da execução do corte e a cláusulas singulares (sem nó OU associado) contendo cortes.

No método Opera, o nó OU atual, no início da execução da primeira alternativa (cláusula ou alternativa de disjunção) com corte, é marcado através de um novo registrador BFREE. Ele é o nó

OU mais recente não inibido. As alternativas, que contem operadores de corte e que não são associadas a um nó OU, são contadas até a criação de um nó OU, utilizando-se de um registrador adicional NCC. A exportação de um nó OU é inibida se ele é posterior ao nó BFREE.

A marcação é efetuada pelas instruções de retrocesso (*try*, *retry* e *trust*) e por uma nova instrução. Tal instrução é gerada no início de uma cláusula, caso a mesma contenha um ou mais operadores de corte. A retirada da marca é realizada pelas instruções de retrocesso e por instrução de corte gerada para o último (versão especial da instrução WAM de corte) corte da cláusula.

Um registrador adicional, SNCC, é usado para o salvamento do contador de cortes (NCC) quando da marcação, esse valor sendo necessário no momento de retrocesso ao primeiro nó inibido.

5. Arquitetura hierárquica

Por limitações de espaço, essa otimização será descrita de forma muito resumida. O protótipo inicial de Opera contem um escalonador centralizado, o qual se adapta perfeitamente à arquitetura da menor configuração da máquina Supernode, denominada Tnode. Nesse protótipo, cada nó contem, além do processo WAM, um processo espião e um processo exportador. O processo espião informa o escalonador da carga da WAM associada (ativa). O escalonador é alocado em um processador central (de controle) com capacidade de comunicação 1 * n por via de comunicação e controle. O processo exportador está sempre à espera de um pedido de exportação por parte do escalonador, sendo responsável pela transferência de contextos. Quando uma WAM inativa solicita uma tarefa ao escalonador, esse analisa o estado das cargas das WAM ativas e escolhe uma entre elas para ser a WAM exportadora, enviando então às duas WAM os respectivos pedidos de importação e exportação.

No entanto, configurações maiores do Supernode, denominadas Meganode, são constituídas de vários Tnode, interligados através de um Transputer de controle adicional, podendo comunicar-se com os Transputer de controle dos Tnode de modo 1 *n por outras vias de controle.

Para tais máquinas, propõe-se uma nova organização do escalonador, hierárquica, onde um processo semelhante (EC) ao do protótipo original é alocado no processador de controle central para fins de escalonamento inter-Tnode, e processos de escalonamento local (EL) são alocados nos Transputer de controle de cada Tnode.

Os processos EL alimentam o EC com informações sobre as suas cargas, basicamente a maior carga de uma máquina ativa local. Tal informação será usada na escolha do Tnode exportador em caso de requisição de importação por um EL sem carga. O processo EC, ao contrário da versão anterior sobre um Tnode, não mantem o estado das seções comuns, necessário à cópia incremental ([4]), pois considera-se que instalações inter-Tnode serão raras, não compensando o custo de controle de seções comuns.

Cada processo EL atende as necessidades de escalonamento (seleção das máquinas exportadora-importadora), como anteriormente, isto é, dando prioridade aos pedidos das máquinas locais. No entanto, em caso de não haver máquina local com capacidade de exportação,

e havendo requisições de importação, essas são repassadas ao processo EC. Além disto, o processo EL deverá atender eventuais requisições de exportação (se possível), encaminhadas pelo EC, a partir de requisições de importação, produzidas por máquinas de outros Tnode, e repassadas ao EC pelos outros EL.

Note-se por exemplo, que um processo EL poderá, ao receber uma autorização de importação inter-Tnode, recusá-la pois alguma de suas máquinas ativas conseguiram, após a decisão de requisitar tal importação, atingir um estado de carga mínimo para exportação. Tais situações aumentam a complexidade dos processos EL, com relação ao escalonador do protótipo sobre um Tnode simples.

6. Conclusão

O protótipo Opera inicial, baseado no modelo pilha de variáveis e datação, apresentou bons ganhos de desempenho, mesmo sem conter diversas otimizações previstas no modelo, como a cópia incremental. A implementação da mesma verificou-se complexa, e com alta sincronização e alguma redundância. Nesse artigo, propos-se uma evolução do modelo, baseada principalmente na cópia de ligações condicionais válidas à WAM importadora e na redução do número de variáveis condicionais. A modificação correspondente do protótipo é necessária, com o intuito de validar essa evolução.

Referências:

[1] Ali, K.A.M. A Method for Implementing Cut in Parallel Execution of Prolog. In Proceedings of 4th Symposium on Logic Programming, San Francisco, pp.449-456, 1987.

[2] Ali, K.A.M. and Karlsson, R. The Muse Or-Parallel Prolog Model and its Performance. In Proceedings of North-American Conference on Logic Programming 90, Austin, pp.757-776, 1990.

[3] Briat, J., Favre, M., Geyer, C., Chassin, J.K. Scheduling of Or-Parallel Prolog on a Scalable, Reconfigurable, Distributed-Memory Multiprocessor. In Proceedings of PARLE'91, 1991.

[4] Geyer, C. Une Contribution à l'Étude du Parallélisme OU en Prolog sur des Machines sans Mémoire Commune. Tese de doutorado. Université Poseph Fourier, Grenoble, 1991.

[5] Sohma, Y et al. A New Parallel Inference Mechanism Based on Sequential Processing. Tecnhical Memorandum No. TM-0131, ICOT, 1985.

[6] Warren, D.H.D. An Abstract Prolog Instruction Set. Tecnhical Note No.309, Artificial Intelligence Center, SRI International, 1983.

[7] Warren, D.H.D. The SRI Model for Or-Parallel Execution of Prolog - Abstract Design and Implementation Issues. In Proceedings of 4th Symposium on Logic Programming, San Francisco, pp.92-102, 1987.